1. the paragraph beginning on page 5, line 19:

FIG. 1 shows a block diagram of an embodiment of the present invention;

A1

FIG. 2 shows a logic diagram in accordance with the present invention; and

FIG. 3 shows an exemplary implementation of the present invention.

2. the paragraph beginning on page 8, line 8:

At step 220, the system invokes a submit form function. In one embodiment, the code depicted in the example of source code listing disclosed below may be used to implement the functionality disclosed herein. At step 230, the system determines whether group members have been previously activated. In one embodiment, this functionality is implemented using a flag that is associated with the unique key. The flag may be saved in a cookie associated with the unique key. This functionality may be implemented as disclosed in the allowSubmit() function of the example of source code listing disclosed below. Other methods of determining whether a member of a functional group has been previously submitted are also possible.

A2

3. The paragraph beginning on page 9, line 13:

If the confirmation attribute is true, activation is allowed and the system may proceed to step 260. If the confirmation attribute is false, the system may proceed to step 290, in which case the form data is not resubmitted. In one embodiment, the system may present an alert message indicating that a related response has been previously submitted and having no control enabling the HTTP request from being resubmitted (e.g., once the user selects OK, no HTTP request is sent to the server). In the embodiment disclosed in the example of source code listing disclosed below, whether the system proceeds to step 260 or 290 is based on the value of a confirmFlag variable, which is based on the confirmation attribute.

A3

4. the paragraph beginning on page 10, line 1:

If the system determines that confirmation is allowed, the system proceeds to step 260. At step 260, the system renders a confirmation message including a control to enable the resubmission of the HTTP request. For example, the system may render a confirm message saying "This form has been previously submitted. Submitting again may result in an error. Do you want to submit?" as depicted in the allowSubmit() function in [Fig. 4] the example of source code listing disclosed below.

Please insert the following on Page 12 after line 14.

An example of a source code listing according to an exemplary embodiment of the present invention is given below. The below example is given for illustrative purposes only, and the embodiments of the present invention may be implemented according to other source code listings known to those skilled in the art.

PageHistory methods

Description:
definition of the PageHistory methods

```
==========================================================
======= */
function PageHistory_setPageFlag(name, flag) {
        //variable used in the cookie: literal string with ' to escape the potential :
inside of name
        var namelit = "'" + name + "'";
        //set it in the cookie string
        //case in it is in pages already, then modify value in cookie
        if (!this.isNotInHistory(name)) {
                var repl = new RegExp(namelit + ":" + "[^\\s]*\\s");
                debug(GenDebug, "this.cookie before", this.cookie);
                debug(GenDebug, "repl", repl);
                this.cookie = this.cookie.replace(repl, namelit + ":" + flag + " ");
                debug(GenDebug, "this.cookie after", this.cookie);
        }
        else {
                //case cookie empty, create it
```

```
                if (this.cookie == "") {
                        this.cookie = "{ " + namelit + ":" + flag + " }";
                }
                //else add it to the beginning
                else {
                        this.cookie = "{ " + namelit + ":" + flag + " , " +
this.cookie.substr(1);
                }
                this.size++;
                debug(GenDebug, "this.size", this.size);
                //case size > maxsize, remove lru = end
                if (this.size > this.maxsize) {
                        debug(GenDebug, "this.cookie before size restraint", this.cookie);
                        this.cookie = this.cookie.slice(0, this.cookie.lastIndexOf(",")) +
"}";
                        debug(GenDebug, "this.cookie after size restraint", this.cookie);
                }
        }
        debug(GenDebug, "this.cookie end", this.cookie);
        //set it in the hashtable
        this.pages[name] = flag;
        //set the cookie
        setCookie("pageh", this.cookie);
}

function PageHistory_getPageFlag(name) {
        if (IsDef(this.pages[name])) return this.pages[name];
}
function PageHistory_isNotInHistory(name) {
        if (IsUndef(this.pages[name])) return true;
        return false;
}

/*=============================================================
========
        PageHistory()

        Description:
            definition of the PageHistory constructor
                        The PageHistory object stores a mru/lru (most and least recently
used) list of size maxSize pages for which you submitted a form
                        When you add a page and the list is full we discard lru and add
it as mru
```

Pages are characterized by a guid and a a flag
The flag indicates if the page has already been submitted
This object has some methods that lets you add a page, update a flag for a page and get a flag from a page.

```
==========================================================================
======= */
//no need to use prototypes since we use only one instance of the object
function PageHistory(maxsize) {
	//define the member variables
	this.maxsize = maxsize;
	this.size = 0;
	//we store a js expression defining an associative array in the cookie
	//we instantiate the array for quick lookups in the js object pages
	//we keep the cookie to manage the lru/mru for serialization
	//this makes the whole thing much faster: remove beginning and add to the
end are very easy ops on strings
	//and we don't have to loop to serialize/deserialize
	this.cookie = getCookie("pageh");
	debug(GenDebug, "this.cookie", this.cookie);
	if (this.cookie == "") {
		this.pages = new Object();
	}
	else {
		this.pages = eval("bozo = " + this.cookie);
		this.size = this.cookie.split(',').length;
		debug(GenDebug, "this.size", this.size);
	}
	//define the methods
	this.setPageFlag = PageHistory_setPageFlag;
	this.getPageFlag = PageHistory_getPageFlag;
	this.isNotInHistory = PageHistory_isNotInHistory;
	debug(GenDebug, "pages", dumpObject(this.pages));
}

var pageHistory = new PageHistory(20);

/*==========================================================================
========
	allowSubmit(name)

	Description:
```

checks if a page is allowed to be submitted. If it is allowed to do so, returns true and sets its flag to 0 so that it won't be allowed again.
to be called on a onSubmit handler

```
================================================================
======= */
function allowSubmit(name, confirmFlag) {
   uKey = pageHistory.getPageFlag(name)
        debug(GenDebug, "uKey", uKey);
   if (uKey == "1") {
                pageHistory.setPageFlag(name, "0");
                return true;
   } else {
        if (confirmFlag) {
           return confirm("This form has previously been submited. Submitting again
may result in an error. Do you want to submit?");
        } else {
           alert("This Form has previously been submited. It cannot be submited
again.");
           return false;
        }
   }
}

/*================================================================
========
        putPageInHistory(name)

        Description:
             puts a page in history if it is not there, with a flag allowing it to be
submitted.
                        To be called at the beginning of your page

================================================================
======= */
function putPageInHistory(name) {
        debug(GenDebug, "pageHistory.isNotInHistory(name)",
pageHistory.isNotInHistory(name));
        if (pageHistory.isNotInHistory(name)) {
                        pageHistory.setPageFlag(name, "1");
        }
}
```

```
function dumpObject(obj) {
	var dump = "";
	for (var i in obj) dump += i + "=" + obj[i] + "\n";
	return dump;
}

/*================================================================
========
	getObject()

	Description: convert object name string or object reference into a valid object
reference
	for both browsers: this is a reference on which you can set some style
attributes
================================================================
======= */
function getObject(obj) {
	var theObj;
	if (typeof obj == "string") {
		var iniObj;
		if (isNav6) {
			iniObj = document.getElementById(obj);
		}
		else {
			iniObj = eval("document." + coll + obj);
		}

		if (IsUndef(iniObj)) {
			return "undefined";
		}

		if (isNav4) {
			return iniObj;
		}
		else {
		// in the IE or NS6 case the iniObj.style object may be undefined
			if (IsDef(iniObj.style)) {
				return iniObj.style;
			}
			else {
				return "undefined";
			}
		}
```

```
	}
	else {
		theObj = obj;
	}
	return theObj;
}

/*=================================================================
========
	getObjectRef()

	Description: convert object name string or object reference into a valid object
reference
	for both browsers, without the style in IE: this is the real object reference
	this function is adapted from Danny Goodman's "Dynamic Html : The
Definitive Reference"
	http://www.amazon.com/exec/obidos/ASIN/1565924940/qid%3D963012863/00
2-0174003-8509633

=================================================================
======= */
function getObjectRef(obj) {//alert("getRef "+obj);

	var theObj;
	if (typeof obj == "string") {
		var iniObj = eval("document." + coll + obj);
//alert("getRef "+iniObj);
		if (IsUndef(iniObj)) {
			return "undefined1";
		}
			return iniObj;

		}
		else {
		theObj = obj;
		}
	return theObj;
}

/*
=================================================================
=======
FUNCTION:	IsUndef
```

```
INPUT:          val - the value to be tested

RETURN:    true, if the value is undefined
           false, otherwise.

PLATFORMS:      Netscape Navigator 3.01 and higher,
                Microsoft Internet Explorer 3.02 and higher,
                Netscape Enterprise Server 3.0,
                Microsoft IIS/ASP 3.0.
================================================================
======= */
function IsUndef( val ) {
    var isValid = false;
     if (val+"" == "undefined")
          isValid = true;

     return isValid;
} // end IsUndef

function IsDef( val ) {
     return !IsUndef(val);
} // end IsUndef

/**
* <pre>
* This function
*      checks if the form is allowed to be submitted.
*      if yes, it sets the action, then calls form.submit()
*
* Usage:
*
*  <code>
*      <a href="Javascript:submitForm('/iMM/somefile.jsp', 'myForm',
'someUniqueKey', true)">
*  </code>
*
* </pre>
*/
function submitForm(action, formName, key, confirmFlag) {
  //we put a provision here to let the users of this function not provide the last
argument, which the ndefaults to true
  if (IsUndef(confirmFlag)) {
```